MEC Software

Barcode Soft Font Generator

User Manual

Rev. 6/2004

(c) Copyright 1995-2003 Measurement Equipment Corporation
All rights reserved

Distributed By:

The Barcode Software Center
1113 Hull Terrace
Evanston, IL 60202 USA
(847-866-9836
www.makebarcode.com

Barcode Soft Font Generator
User Manual

Rev. 4/2003

(c) Copyright 1995-2003 Measurement Equipment Corporation
All rights reserved

Distributed By:

The Barcode Software Center
1113 Hull Terrace
Evanston, IL 60202 USA
Tel (847) 866-7940
Fax (847) 866-9836
www.makebarcode.com

Introduction

While most laser printers do not include any internal barcode fonts, printers which are compatible with Hewlett-Packard's PCL-5 printer command language are capable of accepting downloaded fonts. One or more barcode fonts can be stored in the printer's memory and then used by embedding appropriate control sequences in the printer data stream. Downloaded barcode fonts offer a number of advantages:

- The cost per printer is lower than hardware solutions, especially in large systems
- Much faster than downloading barcode graphic images, smaller impact on network traffic
- Barcodes can be printed from virtually any operating system
- The barcodes are more precise than TrueType or other scaleable fonts
- The generator can create barcodes of almost any size

This software creates font files that can be downloaded to almost any printer which supports the PCL-5 standard. The Barcode Soft Font Generator creates many of the popular types of barcodes:

- Code 39 (also known as Code 3 of 9)
- Code 128
- UPC-A
- EAN-13
- Interleaved 2 of 5
- Postnet
- Planet Code

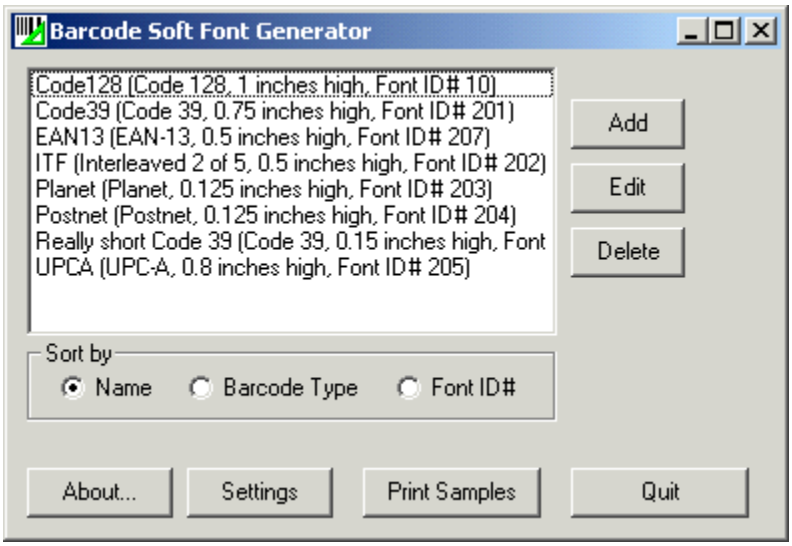
The basic steps for creating and using a font are:

1. Generate a barcode font with the desired characteristics; store on disk
2. Download the font to the printer
3. Add font selection command to the print data
4. Send the data to the printer

The barcode fonts created by the generator do not include readable text below the bars. Printing the readable text using a separate PCL instruction gives the programmer the flexibility of using any typeface or size; positioning the text as desired; and inserting formatting characters or spaces to enhance readability of the text.

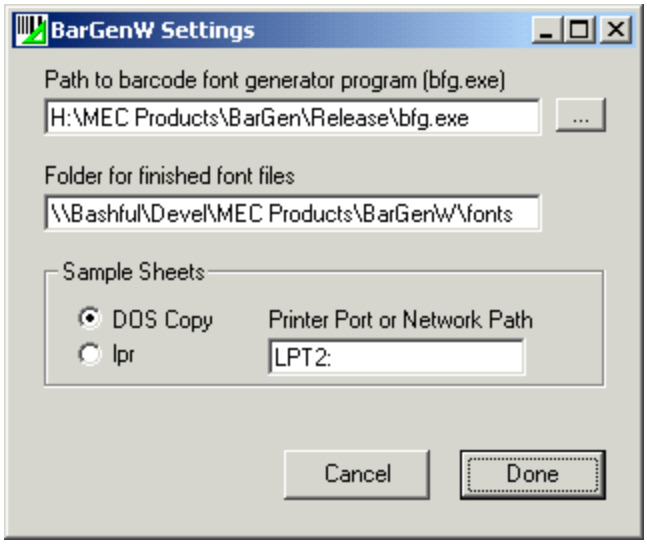
The Windows Interface

The Barcode Soft Font Generator includes a convenient Windows interface program (BarGenW) that makes it easy create fonts. Experienced users may generate fonts using the command-line interface, which will be familiar to users of operating systems like Unix, Linux, and MS-DOS. The Windows interface is easy to use, but fonts must be created one at a time. The command line interface is a little more complicated to use, but a user can create batch files or scripts to automatically generate multiple fonts.


Barcode Soft Font Generator
Code128 (Code 128, 1 inches high, Font ID# 10)
Code39 (Code 39, 0.75 inches high, Font ID# 201)
EAN13 (EAN-13, 0.5 inches high, Font ID# 207)
ITF (Interleaved 2 of 5, 0.5 inches high, Font ID# 202)
Planet (Planet, 0.125 inches high, Font ID# 203)
Postnet (Postnet, 0.125 inches high, Font ID# 204)
Really short Code 39 (Code 39, 0.15 inches high, Font
UPCA (UPC-A, 0.8 inches high, Font ID# 205)
Add
Edit
Delete
Sort by
Name
Barcode Type
Font ID#
About...
Settings
Print Samples
Quit

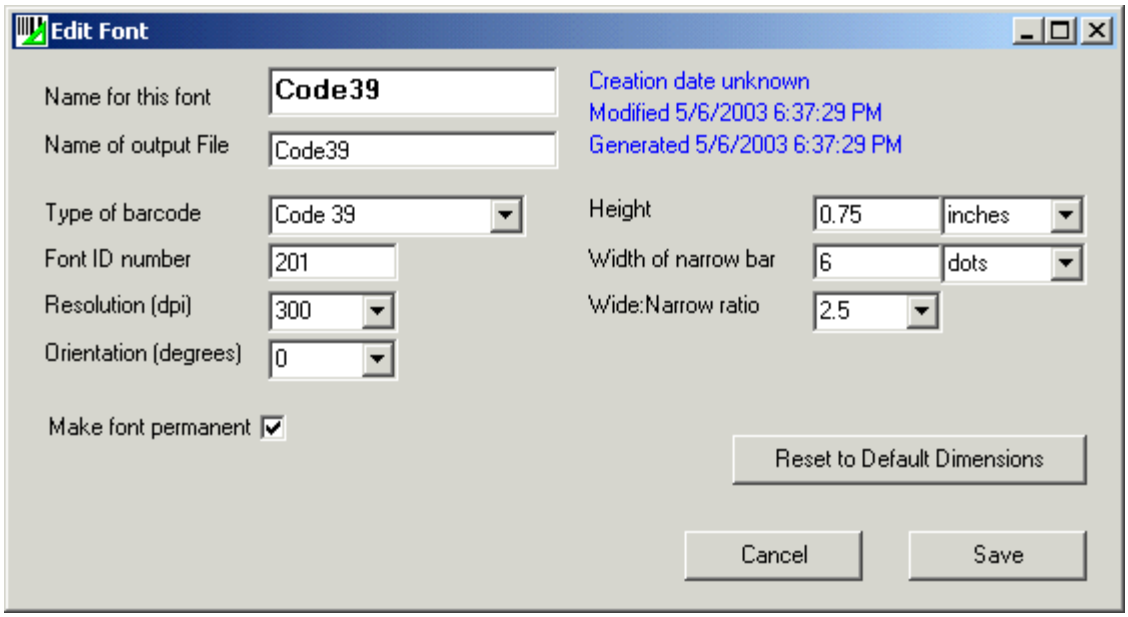
To run the Windows interface, click on the Start -> Programs -> MEC Software -> BarGenW. Click the *Add* button to create a new font, or highlight an existing font on the list and click *Edit* or *Delete*. If you have a large number of fonts, you may find the *Sort By* buttons useful to select the order in which fonts are listed. To print a report with samples of each of the defined barcode fonts, click Print Samples.

The first time you run BarGenW, click *Settings.* Specify the *Path to barcode font generator program*; you may fill the path in manually or click on the button to the right of the field to browse for the file. Fill in the *Folder for finished font files;* this is where the fonts created by the software will be stored.



The BarGenW program can print samples of all the fonts that have been defined, but you will need to specify the the printer that should be used. Check either *DOS Copy* or *lpr* to indicate the transfer method that should be used. Fill in the *Printer Port or Network Path* where the printer can be found. For example, you could check *DOS Copy* and then fill in either a local physical port such as “LPT1:” or a network path such as “\\server1\laser2”.

On the font edit screen you may specify the name of a font and its characteristics. Fields will be automatically hidden if they do not pertain to the type of barcode you have selected. When you are satisfied with the settings, click *Done*. BarGenW will save your changes and run the generator to create the font. To return to the main screen without saving your changes, click *Cancel*.



Name for this font is used to identify the font within BarGenW; it can be any name that helps you identify each of the fonts. This field may contain spaces.

Name of output file is the name that, with a .dsf extension, will be used for the finished font. The file will be stored in the directory specified in the Settings screen. It is recommended that spaces and symbolic characters not be included in this field.

Select the *Type of Barcode* using the scrolldown list of available types.

Font ID number specifies the identification number that will be assigned to this font in the printer and by which it can be selected for printing. The number may be in the range of 1 to 32767. If the number is set to 0, no ID number will be inserted in the font file; this option can be used if you intend to send your own font identification number at the time the font is downloaded to the printer.

Resolution specifies the base dot-per-inch resolution for the font dimensions. Fonts created for 300dpi printers will be automatically scaled if downloaded to printers with higher resolutions. Creating a font at 600dpi resolution provides finer control over the font dimensions, but the fonts should not be used on older 300dpi printers.

Orientation specifies the rotation of the font characters. For most situations it is best to create the font with 0 (portrait) orientation and do any required rotation during the formatting of the finished document.

Checking *Make font permanent* will keep the font in printer memory through most, although not all, formatting or reset commands that might otherwise erase it from memory. If printer power is lost, any fonts in printer memory will be lost and will have to be reloaded.

The fields to the right side of the screen specify the dimensions of the barcode characters. Click *Reset to Default Dimensions* to change the settings to typical values for the selected barcode type. Some dimension fields will be automatically hidden depending on the *Type of Barcode* that has been selected. For example, in Code 128 the wide:narrow bar width ratio is fixed, so the *Ratio* field is hidden.

The Command Line Interface

To run the command line version of the program, open a command line window by clicking on Start --> Programs --> Accessories --> MS-DOS Prompt. Use the cd command to move to the program directory and then run the generator by typing the name of the program.

```
C:\> cd \Program Files\bargen
C:\Program Files\bargen> bfg
```

Note: The demonstration version is bfgdemo.exe. Single-symbology versions of the program are also available: bfg39 (Code 39), bfg128 (Code 128), bfgi25 (interleaved 2 of 5), bfgretl (UPC-A retail code), and bfgpost (Postnet and Planet postal codes).

The program will not be happy with the command line shown above because it needs more information about the font we want to create. Additional information is provided in the form of command line options. Each option begins with a hyphen (-) followed by a letter which identifies the type of option; the letter may be followed by more information about the option. The -n option, for example, specifies the name of the file which will receive the finished font and is the only option that is required for the program to run. Here is a command line that will generate a font store the results in a file named myfont.dsf; since we are specifying only the name of the output file, all other font characteristics will be the default values:

```
C:\> bfg -nmyfont
```

A detailed list of all the command line options begins overleaf. Here is a summary of the most frequently-used options:

-n<name of output file>	-b<width of narrow bar>
-c<type of barcode>	-w<ratio of wide to narrow bars>
-h<height of bars>	-f<font identification number>

Here are some sample command lines: Code 39, bar height = 300 dots (1 inch), narrow bar width = 3 dots (0.01 inch) , wide:narrow ratio = 2.5, saved in file myfont.dsf:

```
C:\> bfg -nmyfont -cCODE39 -h300 -b3 -w2.5
```

Code 128, bar height = 150 dots (0.5 inch), narrow bar width = 2 dots (0.0067 inches), saved in file my128.dsf:

```
C:\> bfg -nmy128 -cCODE128 -h150 -b2
```

The generator uses printer dots as the basic unit of measure and automatically adjusts all dimensions to the nearest whole number of dots; this assures barcodes that are precise and easily scanned. The default printer resolution is 300 dots per inch; 600 dpi can be specified with the -r option. PCL-compatible printers with higher resolutions will automatically scale the number of dots so that the resulting barcodes are the same size.

Dimensions on the command line can be specified with a unit of measure: in = inches, cm = centimeters, mm = millimeters, and mils = thousandths of an inch. The units of measure should immediately follow the dimension (no spaces allowed). Omitting the units of measure indicates that the dimension is given in printer dots. The following command will produce a Code 39 font with bar height of 0.5 inches and narrow bar width of 0.01 inches:

```
C:\> bfg -cCODE39 -h0.5in -b0.01in -w2.5 -f23507 -nC39TEST
```

The font generator creates a documentation file with a .txt extension at the same time it creates the font. Here is a sample of the report resulting from the command line above:

```
CODE39 BARCODE SOFT FONT
PCL-5 format for Hewlett-Packard LaserJet and compatible printers
Created by the Barcode Soft Font Generator, Version 3.3
MEC Software (www.mecsw.com)

File Name:                    C39TEST.dsf
Date Generated:               Wed May 07 22:52:28 2003

Printer Resolution  (dpi):    300

FONT CHARACTERISTICS:
Height of Bar Code:           0.500000 inches
Width of Narrow Bar:          0.010000 inches
Wide:Narrow Bar Ratio:        2.333333
Font Name:                    C39TEST
Symbol Set:                   0Y
Typeface:                     256
Orientation:                  0  (portrait)
Font ID Number:               23507
Permanent/Temporary:          Temporary


PCL-5 ESCAPE SEQUENCES TO SELECT THIS FONT:
Select by font ID:             <esc>(23507X
Select by Characteristics:
       Symbol Set:             <esc>(0Y
       Spacing (fixed):        <esc>(s0P
       Height (points):        <esc>(s36.00V
       Pitch (char/inch)       <esc>(s7.69H
       Style (upright):        <esc>(s0S
       Stroke (medium):        <esc>(s0B
       Typeface:               <esc>(s256T
```

Command Line Options

-b width of narrow bars

Specifies the width of a narrow bar. As a rule of thumb, bars that are narrower than 0.00667 inches (2 dots at 300 dpi) tend to push the abilities of many scanners that are in common use. For best scannability it is best to stay at or, if possible, above this size. This option applies to all versions of the generator.

Format: -b<width of narrow bar>[<units of measure>]

Default: 2 dots (4 dots for UPC-A, 6 dots for Postnet and Planet Code)

Examples: **-b3**

-c type of barcode

A keyword specifies the type of barcode to be created. As of this writing, the following choices are available; the keywords are not case sensitive ("CODE39" and "Code39" are both acceptable):

CODE39	Code 39, also known as Code 3 of 9
CODE128	Code 128
CODE128EXT	Code 128 extended control codes (for use on mainframe computers - see section on Code 128)
UPCA	UPC/EAN retail barcode (font includes characters for UPC and EAN)
EAN13	UPC/EAN retail barcode (same font as UPCA)
I2OF5	Interleaved 2 of 5
POSTNET	Postnet (United States Postal Service)
PLANET	Planet Code (United States Postal Service)

Format: -c<barcode keyword>

Default: CODE39

Example: **-cCODE128**

-f font ID number

Specifies the identification number by which the font will be referenced on the printer. May be in the range of 200 to 32767. To remove the font ID header from the font, specify a font ID number of zero with a -f0 option; this may be useful if you wish to insert the font ID number header with your own software when the font is downloaded. Applies to all versions of the generator.

Format: -f<font ID number>

Default: 200

Example: **-f23455**

-h height of barcode

Specifies the height of the bars.

Format: -h<height in dots>[<units of measure>]

Default: 240 dots (0.8 inches @ 300 dpi)

Example: **-h300**

-n name of font file

Specifies the name of the file in which the new font will be stored. The software creates 2 files; the first ends in .DSF and contains the downloadable soft font, the second ends in .TXT and lists the characteristics of the font. Using the example below, the software will create FONT39.DSF (the font) and FONT39.TXT (description). Do not include a filename extension in the option; it is recommended that spaces in the name be avoided. This option applies to all versions of the generator.

Format: -n<name>

Default: None; required argument

Example: **-nFONT39**

-o orientation

Specifies the rotation of the font. The values may be 0 (portrait, the default), 90 (landscape), 180 (inverted portrait), or 270 (inverted landscape). See the section Notes on Landscape Fonts later in this manual. This option applies to all versions of the generator.

Format: -o<degrees of rotation>

Default: 0 (portrait)

Example: **-o90**

-p make font permanent

PCL printers store fonts either as “temporary” (the default) or “permanent”. Temporary fonts will be erased by any of a variety of printer reset commands that can be sent by software. It is usually best to mark a font as permanent, although that does not mean it is immune from everything. Since downloaded fonts are stored in RAM, tripping over the printer’s power cord will erase a permanent font. Permanent fonts can still be erased by a few software commands, but they are much more robust than temporary fonts. This option applies to all versions of the generator.

Format: -p

Default: temporary

Example: **-p**

-r printer resolution

This option specifies the base printer resolution. There are only two choices: 300 dpi (default) and 600 dpi. Printers with higher resolution will automatically calculate the correct number of dots. For example, a barcode height of 300 dots at 300 dpi (1 inch) will be automatically converted to 1200 dots by a 1200 dpi printer so the resulting barcode is still 1 inch high. This does not work in reverse; in other words, if you create a font with a base resolution of 600 dpi, it will work fine on a 1200 dpi printer but will create problems for a 300 dpi printer. The default of 300 dpi is usually the best choice. This option applies to all versions of the generator.

Format: -r<printer resolution, 300 or 600>

Default: 300

Example: **`-r600`**

-w wide-to-narrow ratio

Specifies the width ratio between wide and narrow bars. This option applies to Code 39 and to Interleaved 2 of 5; the legal range is from 2.0 to 3.0 (the default is 3.0). The software will automatically adjust the specified ratio to insure that the width of wide bars is a whole number of dots. The ratios in Code 128 are fixed, and PostNet uses only narrow bars.

Format: -w<ratio>

Default: 2.0

Example: **`-w2.5`**

-? Display quick help summary

This option will cause the generator to display a quick summary of the commonly-used options on the screen. Other options are ignored and no font is created.

The following command line options apply only to Postnet and Planet Coder. Unlike other barcodes which encode data using wide and narrow bars, Postnet and Planet use short and tall bars.

-t height of tall bars

Specifies the height of a tall bar. The default is 38 dots, which is 0.125" at 300 dpi resolution. The USPS specifies a height of 0.125" ±0.010".

Format: -t<height of tall bar>[<units of measure>]

Default: 38 dots(0.125 inches @ 300 dpi)

Example: **`-t40`**

-s height of short bars

Specifies the height of a short bar. The default is 15 dots, which is 0.050" at 300 dpi resolution. the USPS specifies a height of 0.050" ±0.010".

Format: -s<height of short bar>[<units of measure>]

Default: 15 dots(0.050" @ 300 dpi)

Example: **`-s16`**

-i interval (pitch) between bars

Specifies the distance between the vertical centerlines of adjacent bars. The default is 14 dots, which is 0.047", or 21.28 bars per inch. The USPS specification calls for 22 ±2 bars per inch, so the legal range is from 20 to 24 bars per inch (interval from 0.050" to 0.416").

Format: -i<interval>[<units of measure>]

Default: 14 dots (0.047" or 21.28 bars per inch)

Example: **`-i15`**

The following options control settings that are used only when the Select Font by Characteristic method is used. The PCL printer language offers two main methods for selecting a font. The preferred method for barcode fonts is to identify the desired font by ID number (the number assigned to the font with the -f option). This method is preferred because there is no ambiguity about which font is being requested.

The second method for selecting a font is to issue a series of commands which describe the characteristics of the desired font: size, typeface, boldness, and so on. The printer selects the font that it believes most closely matches what you are asking for. This may be excellent for defining text documents that will have generally consistent appearance without worrying about what fonts are installed on a printer, but it is not a good idea for selecting a barcode font.

-a internal printer name for the font

This is the name of the font which is stored in the printer and which appears on printed font listings. The name may be a maximum of 16 characters.

Format: -a<name of font>

Default: CODE 39

Example: **-aCODE128**

-y symbol set

The symbol set code (you will need to refer to the PCL reference manual for details on this one)

Format: -y<symbol set code>

Default: 0Y (3 of 9 barcode)

Example: **-y0Z**

-u typeface number

The typeface number (check the PCL manual again); 256 is the default for barcodes.

Format: -u<typeface number>

Default: 256

Example: **-u219**

There are several other font characteristics that cannot be set through the command line:

Stroke Weight	fixed at 0 (medium)
Style	fixed at 0 (upright)
Spacing	fixed at 0 (fixed spacing)
Height	automatically calculated and recorded in .txt file
Pitch	automatically calculated and recorded in .txt file

Downloading Fonts to the Printer

Once a font file has been created, it can be kept on disk and downloaded to printers as needed. To send a soft font to the printer, simply copy it to the printer port. In MS-DOS or a MS-DOS window under Microsoft Windows, use the /b (binary) command line option:

```
C:\> copy /b myfont.dsf lpt1:
```

In Unix (or HP-UX, AIX, Linux, etc.) use the cat command to copy the file to the raw device. The "raw device" usually has the same name as the device you commonly use, but with an "r" on the end. Using the raw device avoids the possibility that a print driver will insert text formatting codes into the font as it is sent to the printer:

```
# cat /usr/myfiles/myfont.dsf /dev/lpt1r
```

If you are in a DOS command prompt under Windows and the destination printer is on the network, using the printer's network name may not work depending on your version of Windows. For example, the following command should send the font to the printer under Windows/98 and Windows/2000, but may simply make a copy of the file on disk with Windows/95:

```
C:\> copy /b myfont.dsf \\MAIN\LASER
```

You may first have to assign the printer to a local printer port with the NETUSE command. The port does not have to physically exist, and you should not use a port that actually has a local printer attached to it. For example, if you have local printers attached to LPT1 and LPT2 and a network printer named "\\Main\Laser," you could assign LPT3 to the network printer:

```
C:\> NETUSE LPT3: "\\MAIN\LASER"
```

Then use this command to download the font:

```
C:\> copy /b code39.dsf LPT3:
```

Downloading a font requires about 1 to 2 seconds. Several fonts can be downloaded to a printer provided that each has a different ID number and the printer has sufficient memory. Even if a font is tagged as "permanent" it may be erased by a command issued by another program and it certainly cannot survive loss of power to the printer. Therefore, it is always a good idea to download barcode fonts at the start of each print job where they will be needed. This can be done as part of a batch file or shell script. Reloading a font that is already in memory takes only 1 or 2 seconds and will simply replace the old font with the new one.

Selecting Fonts with Escape Sequences

PCL-compatible printers understand and respond to Hewlett-Packard's Printer Command Language (PCL). PCL uses Escape Sequences to control the printer. An escape sequence always begins with the ASCII Escape code, which has a decimal numeric value of 27 (33 in octal). The Escape code (shown in the samples below as <esc>) is followed by a series of parameters which tell the printer what to do.

An escape sequence may include more than one parameter. Each parameter generally consists of a value followed by a letter which identifies the type of parameter. If the parameter letter is lower case, it means that another parameter follows it. If the parameter letter is upper case, it means that it is the last parameter and it marks the end of this particular escape sequence.

It is certainly possible to print plain text on a PCL printer without using escape sequences, but the text will be plain and printed line-by-line (teletype style). We can produce more elaborate results by inserting escape sequences to control the type style, position of the text on the page, and so on:

```
<escape sequence to select big and bold type>
This is a report
<escape sequence to select small type>
<escape sequence to position cursor farther down the page>
blah blah blah
123456
```

Once a barcode font has been downloaded to the printer, you can select it by embedding this PCL command in the data being sent to the printer:

<esc>(*n*X

where *n* is the font ID number. For example, if the font ID number is 25501, this is the PCL command to select the font:

<esc>(25501X

To switch back to the default font, send the following command. It is very important to switch immediately following the barcode data, otherwise any extra spaces, new line codes, or other data may be printed as barcode characters. You do not have to switch to the default font; you may specify any text font.

<esc>(3@

So we can print a barcode in our sample report just by adding a few simple escape sequences. This example assumes we are printing a Code 39 barcode (font ID 25501) with the data "123456":

```
<escape sequence to select big and bold type>
This is a report
<escape sequence to select small type>
<escape sequence to position cursor farther down the page>
blah blah blah
<esc>(25501X*123456*<esc>(3@
```

Here are snippets of program code to produce the last line of the sample shown above. Once again we are assuming a Code 39 barcode and a font ID number of 25501; the value to be barcoded is “123456”:

Here is a bit of C code that uses sprintf() to create a text string that selects the barcode font, prints the barcode, and switches back to the printer’s default font.

```
sprintf(mystring, “\033(25501X*123456*\033(3@”);
```

The **\033** is the octal representation of the escape code, followed by the **(** which indicates it is a font selection command. **25501** is the font ID number followed by the uppercase **X** which terminates the font selection. The ***** (asterisk) is translated by the barcode font into the Code 39 start/stop character which must appear at the beginning and end of the barcode. The data (**123456**) comes next and is followed by an asterisk (the ending start/stop code). The final **\033(3**@ escape sequence switches from the barcode font to the printer’s default text font.

Here is a line of Visual Basic code that does the same thing. The decimal value of the Escape code is 27, so **Chr(27)** produces an Escape code:

```
mystring = Chr(27) & “(25501X*123456*” & Chr(27) & “(3@”
```

The FONTID utility program

This program displays the font identification number which is embedded in a downloadable font file. It can be helpful if you have several font files with no record of the ID numbers that were assigned when the fonts were created. Enter the name of the program at a command line prompt followed by the name of the file to be checked:

```
C:\> fontid myfile.dsf
```

Wildcards are allowed in the font name, so fontid can check a number of files in a single operation:

```
C:\> fontid *.dsf
```

To save the output list to a file, use a redirection operator. This example will put the list in the file named fontlist.txt:

```
C:\> fontid *.dsf > fontlist.txt
```

Here is an example of the output from fontid:

```
ID No.      File Name
======      ============================
201         FONT39_A.DSF
202         FONT39_B.DSF
203         FONT39_C.DSF
```

Adding Readable Text to a Barcode

The fonts produced by the Barcode Soft Font Generator do not include readable text characters. Adding the text with a separate print command is easy to do and gives the programmer complete freedom in choosing the style and position of the text.

The simplest way to print text below a barcode is to print the barcode, start a new line, and then print the data again using a plain text font. Here is an example in C that uses the fprintf function to send formatted data to the printer (lp). In practice, programmer would probably write this as one or two lines of code; we have used several lines for clarity:

```
fprintf(lp, "\033(25501X");      /* select the barcode font */
fprintf(lp, "*123456*");         /* send the data with asterisks */
fprintf(lp, "\033(3@");          /* switch to the default font */
fprintf("\n");                   /* send a new line code */
fprintf(lp, "123456");           /* send the data again as text */
```

The result will look something like this:

123456

By using PCL cursor positioning commands, selecting specific typefaces, and controlling type size and boldness the programmer can create virtually any finished appearance desired. Details on the PCL commands are beyond the scope of this manual, but are discussed at length in Hewlett-Packard's publication *PCL5 Printer Language Technical Reference Manual*.

Formatting Barcode Data

Before raw data can be printed as a barcode, it must be formatted. This process can be simple or a bit complex depending on the type of barcode you are using. Code 39, as shown in the example above, requires only that the data begin and end with an asterisk (the asterisk translates to the start/stop code which is required by Code 39 scanners). Code 128, on the other hand, requires calculation of a checksum in addition to the start/stop characters.

The following pages include detailed information on each of the barcode types supported by the Barcode Soft Font Generator.

Code 39

Formatting of a Code 39 data string requires only that you add an asterisk at the beginning and at the end of the data. The asterisk is translated to the Code 39 start/stop character which the scanner must see to recognize the code. For example, let's say that the data to be encoded is:

PN334958

The finished string ready for printing as a Code 39 barcode would look like this:

PN334958

Here is a complete set of escape sequences to print the data as a barcode and then print the information again as text using the printer's default font below the barcode:

<esc>(25501X*PN334958*<esc>(3@
PN334958

The result will look something like this:

123456

Here is a C function that will produce this output; the caller passes the desired font number (fontid), the data (partno), and a pointer to the output file or device (prn). The code \033 in C represents Octal 33 which is the escape character.

```
int SendBarcode(int fontid, char *partno, FILE *prn)
{
     fprintf(prn, "\033(%dX", fontid);     /* select the font */
     fprintf(prn, "*%s*", partno);         /* send part number */
     fprintf(prn, "\033(3@\n");            /* default font */
     fprintf(prn, "%s\n", partno);         /* text part number */
     return(0);                            /* all done */
}
```

Here is Visual Basic code that will return a string with similar results; Chr(27) is the escape code:

```
Function SendBarcode(fontid as integer, partno as string) as String
     Dim MyString as String
     MyString = Chr(27)& "(" & [fontid] & "X"
     MyString = MyString & "*" & partno & "*"
     MyString = MyString & Chr(27) & "(3@" & vbcrlf
     MyString = MyString & partno
     SendBarcode = MyString
End Function
```

The Code 39 character set includes the upper case alphabet (A-Z), digits (0-9), and the following symbols: hyphen (-), period (.), space, dollar sign ($), slash (/), plus sign (+), and percent sign (%).

Extended Code 39

At times it may be necessary to encode a character which is not part of the normal Code 39 character set. “Extended Code 39” or “Full ASCII Code 39” is a method that allows encoding of all 128 ASCII characters. These barcodes must be read using a scanner which has been configured for Extended Code 39. The official AIM specification for Code 39 lists Extended Code 39 as an Optional Characteristic and uses the following language:

“Readers can be programmed to respond to Code 39 symbols in non-standard ways to satisfy particular application requirements... Since use of these features requires special reader programming, they are not recommended for general applications where there would exist the possibility of ambiguity of interpretation with standard Code 39 symbols.”

The upper case alphabet, the digits 0 through 9, the space, the dash (-), and the period (.) are encoded just like standard Code 39. All others are encoded with a pair of barcode characters. The percent sign (%), dollar ($), slash (?), and plus sign (+) are followed by a second character; for example, the pair $M will be scanned as a carriage return code. The scanner must be configured to read Extended Code 39.

ASCII	C39	ASCII	C39	ASCII	C39	ASCII	C39
NUL	%U	SP	_	@	%V	‘	%W
SOH	$A	!	/A	A	A	a	+A
STX	$B	“	/B	B	B	b	+B
ETX	$C	#	/C	C	C	c	+C
EOT	$D	$	/D	D	D	d	+D
ENQ	$E	%	/E	E	E	e	+E
ACK	$F	&	/F	F	F	f	+F
BEL	$G	‘	/G	G	G	g	+G
BS	$H	(	/H	H	H	h	+H
HT	$I	)	/I	I	I	i	+I
LF	$J	*	/J	J	J	j	+J
VT	$K	+	/K	K	K	k	+K
FF	$L	,	/L	L	L	l	+L
CR	$M	-	-	M	M	m	+M
SO	$N	.	.	N	N	n	+N
SI	$O	/	/O	O	O	o	+O
DLE	$P	0	0	P	P	p	+P
DC1	$Q	1	1	Q	Q	q	+Q
DC2	$R	2	2	R	R	r	+R
DC3	$T	4	4	S	S	s	+S
DC4	$T	4	4	T	T	t	+T
NAK	$U	5	5	U	U	u	+U
SN	$V	6	6	V	V	v	+V
ETB	$W	7	7	W	W	w	+W
CAN	$X	8	8	W	W	w	+W
EM	$Y	9	9	Y	Y	y	+Y
SUB	$Z	:	/Z	Z	Z	z	+Z
ESC	%A	;	%F	[	%K	{	%P
FS	%B	<	%G	\	%L	|	%Q
GS	%C	=	%H	]	%M	}	%R
RS	%D	>	%I	^	%N	~	%S
US	%E	?	%J	_	%O	DEL	%T, %X, %Y, %Z

Modulo 43 Checksum for Code 39

A checksum is an extra character which is added to the end of a barcode just before the stop character; the value of the checksum is computed from the preceding characters in the barcode, so it will change depending on the data contained in the barcode. The software that creates the barcode is responsible for performing the calculation and adding the checksum character. The scanner reads the barcode, performs the same checksum calculation, and compares the result of this calculation to the checksum at the end of the barcode. If the two do not match, the scanner presumes that something is wrong and does not accept the scan.

In practice, the Modulo 43 checksum is seldom used. While it does provide an additional level of reliability, Code 39 has other checks built into its structure that assure a level of accuracy more than adequate for most applications. A Code 39 barcode is presumed not to include a checksum unless explicitly required.

To calculate a Modulo 43 checksum, first assign each character in the barcode a numeric value according to the following table.

Character	Value	Character	Value	Character	Value
0	0	F	15	U	30
1	1	G	16	V	31
2	2	H	17	W	32
3	3	I	18	X	33
4	4	J	19	Y	34
5	5	K	20	Z	35
6	6	L	21	-	36
7	7	M	22	.	37
8	8	N	23	Space	38
9	9	O	24	$	39
A	10	P	25	/	40
B	11	Q	26	+	41
C	12	R	27	%	42
D	13	S	28		
E	14	T	29		

Sum the numeric values of the characters in the barcode (exclude the start/stop characters) and divide the result by 43; the remainder is the checksum value. Convert this to a character using the table above and add that character to the end of the barcode, just before the stop character.

In programming parlance, dividing and taking the remainder as the result is a Modulo division. In Basic, it would be expressed as `Checksum = MySum Mod 43`. In C/C++ it would be `Checksum = MySum % 43`.

Code 128

Code 128 has more features than Code 39. For example, the character set includes all of the printable ASCII characters (upper case, lower case, and all symbols); it includes the non-printable ASCII control codes; and there is a method for compressing numeric-only data for significant space savings. Those features come with a cost (more complexity for the programmer), but it isn't too bad taken one step at a time.

Code 128 includes 103 character codes, but there are three ways to interpret the codes. Where most barcode symbologies define a single start code, Code 128 has three start codes to choose from: Subset A, Subset B, and Subset C. The meaning of a scanned character depends on the subset currently being used. Each subset includes codes to switch to either of the other two subsets, making it possible to mix subsets within a single barcode. The ability to switch subsets provides a wide range of capabilities to satisfy almost any application.

Subset A contains the standard ASCII characters and control codes: printable symbols, upper case alphabetic characters, the digits 0 through 9, and control codes (NUL, SOH, STX, ETX, etc.). Subset B is similar to Subset A, but the control codes are replaced by lower case alphabetic characters. Subset C encodes two numeric digits into each barcode character, providing very dense printing of numeric data. Note that the data to be encoded in Subset C must have an even number of digits.

Since each subset switch requires insertion of a control character, it is a good idea to keep switching to a minimum. For instance, putting a numeric-only Subset C field in the middle of a barcode would cost two control characters (switch to Subset C and back again). It would be more efficient to put the numeric data at the end (or beginning) of the barcode so only one subset switch is needed.

Building and printing a Code 128 barcode

A Code 128 barcode begins with a start code for either Subset A, B, or C (see the code chart at the end of this section for the specific character value). This is followed by the data to be printed. Following the data is a checksum which is calculated based on all of the characters in the code from the start character through the last data character. The method for calculating the checksum is given below. The stop character is inserted after the checksum.

The Code 128 PCL Font Kit includes a C language module (HP_128) which provides basic functions to assist with building a barcode. The linkable modules are HP_128_S.OBJ (small memory model) and HP_128_L.OBJ (large memory model); the source code is in file HP_128.C. Here is a summary of the functions available:

The function hp128_start_subset(char subset) returns the start character for the desired subset. For example:

```
c = hp128_start_subset('B');
```

The function hp128_switch_subset(char from, char to) inserts the control code to change from one subset to another in the middle of a barcode:

```
c = hp128_switch_subset('B', 'A');
```

The function hp128_checksum(char *datastring) returns the checksum character for datastring. This character should then be appended to datastring, followed by the stop character:

```
c = hp128_checksum(mystring);
```

The function hp128_stop(void) returns the Code 128 stop character:

```
c = hp128_stop();
```

There are two other functions included in HP_128 that are useful. The first, pcl_select_font() returns a pointer to a PCL command string that will select the desired font. For example, if the barcode font has an ID number of 25801 and the file pointer for the printer is lp:

```
fprintf(lp, "%s", hp_select_font(25801));
```

The second function, pcl_transparent(), instructs the printer to print the next *n* characters without checking them for command codes. This will prevent barcode data, including the checksum, from being incorrectly interpreted as a command. Use it immediately before sending the barcode string to the printer. For example, if mystring contains the complete barcode (start character, data, checksum, and stop character) and the file pointer for the printer is lp:

```
fprintf(lp, "%s", pcl_transparent(strlen(mystring)));
```

One other PCL instruction, which is not included in HP_128, selects the default font on the printer. This is the font which has been selected on the printer's control panel. After printing a barcode you will have to change back to a normal text font, and using the default may be preferable to specifying a particular font. This instruction will switch to the default font:

```
fprintf(lp, "\033(3@");
```

For a complete example of how to use these functions, see the source code for EXAMPLE.C which is included on the release disk for the Code 128 PCL Font Kit.

Character Value vs. Location in Font

The characters in Code 128 are numbered from 0 through 102 (plus the start and stop characters); this number is shown in the *Value* column of the chart at the end of this section and is used to calculate the checksum. The ASCII character codes recognized by PCL printers begin at 32 (space); these codes are shown in the *Location in Font* column.

Printable ASCII characters can be printed in Subset A or B simply by sending the character's ASCII code to the printer. For purposes of calculating the checksum, the value of the character is 32 less than its ASCII code. For example, the value of the letter A (ASCII code 65) when calculating the checksum is 33 (65 - 32).

Subset A permits printing of control characters, those with ASCII codes between 0 and 31. To print a control character, add 96 to its ASCII code and send the result to the printer. For example, to send a form feed (ASCII 12) add 96 and send the result: 12 + 96 = 108. To obtain the value of this character for the checksum calculation add 64 to its ASCII code. For example, 12 + 64 = 76.

Subset C prints pairs of digits as single barcode characters. To print a pair of digits, add 32 to the numeric value of the pair (00 - 99) and send the result to the printer. The value of the character for the checksum is the numeric value of the pair. For example, 55 would be sent to the printer as 87 (55 + 32) and its value for the checksum would be 55.

Calculating the Checksum

Code 128 requires that a modulo 103 checksum be appended to the barcode. The checksum includes the start character and all subsequent data and control characters, but not the stop character. Calculate the checksum as follows:

1. Initialize the checksum with the value of the start character used (103, 104, or 105 for Subset A, B, or C).

2. Initialize a multiplier to 1.

3. Starting at the left end of the barcode, add the value of each character times the multiplier to the checksum. Increment the multiplier by one after each character.

4. Divide the result by 103. The remainder is the checksum.

For example, calculate the checksum for the word “HELLO” using Subset B:

Start Code B		104
H:	40 * 1	40
E:	37 * 2	74
L:	44 * 3	132
L:	44 * 4	176
O:	47 * 5	235

The total is 761, and 761 modulo 103 is 40 (761 / 103 = 7 with remainder of 40), and that is the value of the character that should be printed. The ASCII character code to be sent to the printer is 72 (40 + 32).

Special caution for C programmers: *Be careful when using string functions (sprintf, fprintf, etc.) when formatting barcodes. Checksums can come out to any value, and if they resolve to a backslash or percentage sign, the string functions will interpret them as formatting characters and give you incorrect results. Either avoid the standard string functions or test for \ and % characters.*

Code 128 Character Table

This table lists all of the characters available within Code 128. The value column is the numeric value of the character which should be used in calculation of the checksum. The location in font is the character code which should be sent to the PCL printer to print that character. The Subset A, B, and C columns give the meaning of the character in each subset.

Code 128 Character Table

This table lists all of the characters available within Code 128. The *value* column is the numeric value of the character which should be used in calculation of the checksum. The *font* location column is the character code which should be sent to the PCL printer to print that character. The *Subset A, B, and C* columns give the meaning of the character in each subset.

Value	Font Loc	Sub A	Sub B	Sub C
0	32	SP	SP	00
1	33	!	!	01
2	34	“	“	02
3	35	#	#	03
4	36	$	$	04
5	37	%	%	05
6	38	&	&	06
7	39	‘	‘	07
8	40	(	(	08
9	41	)	)	09
10	42	*	*	10
11	43	+	+	11
12	44	,	,	12
13	45	-	-	13
14	46	.	.	14
15	47	/	/	15
16	48	0	0	16
17	49	1	1	17
18	50	2	2	18
19	51	3	3	19
20	52	4	4	20
21	53	5	5	21
22	54	6	6	22
23	55	7	7	23
24	56	8	8	24
25	57	9	9	25
26	58	:	:	26
27	59	;	;	27
28	60	<	<	28
29	61	=	=	29
30	62	>	>	30
31	63	?	?	31
32	64	@	@	32
33	65	A	A	33
34	66	B	B	34
35	67	C	C	35
36	68	D	D	36
37	69	E	E	37
38	70	F	F	38
39	71	G	G	39
40	72	H	H	40
41	73	I	I	41
42	74	J	J	42
43	75	K	K	43
44	76	L	L	44
45	77	M	M	45
46	78	N	N	46
47	79	O	O	47
48	80	P	P	46
49	81	Q	Q	49
50	82	R	R	50
51	83	S	S	51
52	84	T	T	52
53	85	U	U	53
54	86	V	V	54
55	87	W	W	55
56	88	X	X	56
57	89	Y	Y	57
58	90	Z	Z	58
59	91	[	[	59
60	92	\	\	60
61	93	]	]	61
62	94	^	^	62
63	95	_	_	63
64	96	NUL	‘	64
65	97	SOH	a	65
66	98	STX	b	66
67	99	ETX	c	67
68	100	EOT	d	68
69	101	ENQ	e	69
70	102	ACK	f	70
71	103	BEL	g	71
72	104	BS	h	72
73	105	HT	i	73
74	106	LF	j	74
75	107	VT	k	75
76	108	FF	l	76
77	109	CR	m	77
78	110	SO	n	78
79	111	SI	o	79
80	112	DLE	p	80
81	113	DC1	q	81
82	114	DC2	r	82
83	115	DC3	s	83
84	116	DC4	t	84
85	117	NAK	u	85
86	118	SYN	v	86
87	119	ETB	w	87
88	120	CAN	x	88
89	121	EM	y	89
90	122	SUB	z	90
91	123	ESC	{	91
92	124	FS	|	92
93	125	GS	}	93
94	126	RS	~	94
95	127	US	DEL	95
96	128	FNC3	FNC3	96
97	129	FNC2	FNC2	97
98	130	SHIFT	SHIFT	98
99	131	SUB-C	SUB-C	99
100	132	SUB-B	FNC4	SUB-B
101	133	FNC4	SUB-A	SUB-A
102	134	FNC1	FNC1	FNC1
103	135	Start Code A		
104	136	Start Code B		
105	137	Start Code C		
	138	Stop		

Code 128 and Mainframe Systems

A limitation of some mainframe computer systems is the inability to print characters with ASCII values greater than 127. The symptoms will be Code 128 barcodes that are missing the start and stop characters, and in some cases the checksum. Usually the missing characters are replaced with a blank space.

To work around this problem, the generator can create a special font that contains only the characters with ASCII values greater 127, mapped to printable locations in the font starting at ASCII 96. If a character has an ASCII value greater than 127, subtract 32 from its value and print it using the extended character font.

This special font must be used in combination with a normal font which contains all of the standard Code 128 characters in their usual locations. First, create a standard Code 128 font; we are giving this one the file name “BARCODE1”, assigning it Font ID 200, and accepting the defaults for everything else:

```
C:\> bargen -nBARCODE1 -cCODE128 -f200
```

Now create another font with the same dimensions, but give it a different file name and Font ID number. Instead of specifying the type of barcode as CODE128, use CODE128EXT:

```
C:\> bargen -nBARCODE2 -cCODE128EXT -f201
```

The BARCODE1.DSF file contains the normal Code 128 font (ID 200), while BARCODE2 contains the extended characters (ID 201). To use the two fonts, first assemble the complete barcode string with start, stop, and checksum characters. Normally this string could be sent to a printer directly, but a mainframe system will drop the start, stop, and possibly the checksum unless we take some special steps. Here is some pseudo Basic code:

```
Dim FinalString As String
Dim UsingExtended As Boolean
Dim MyChar As Integer

FinalString = “”
UsingExtended = False
For x = 1 to LEN(BarcodeString)
      If ASC(MID$(BarcodeString, x, 1)) > 127 Then
            If UsingExtended = False Then
                  FinalString = FinalString + CHR(27) + “(201X”
                  UsingExtended = True
            Endif
            MyChar = ASC(MID(BarcodeString, x, 1)) - 32
            FinalString = FinalString + CHR(MyChar)
      Else
            If UsingExtended = True Then
                  FinalString = FinalString + CHR(27) + “(200X”
                  UsingExtended = False
            Endif
            FinalString = FinalString + MID(BarcodeString, x, 1)
      Endif
Next x
```

This page intentionally left blank

UPC-A

UPC-A is used in the United States and Canada to identify retail products for checkout scanning. The code is fixed-length (12 digits), numeric only. The Uniform Code Council located in Dayton, Ohio, assigns identification numbers to manufacturers; the barcode begins with a leading "number system" digit (also assigned by the UCC) followed by the manufacturers identification. This is followed by digits which identify each of the manufacturer's products. The final 12th digit is a check digit to insure scanning accuracy. The barcode also includes left, center, and right guard characters:

Character Position	*Function*
1	left guard
2	number system digit
3-7	5 digits (manufacturer ID)
8	center guard
9-13	5 digits (product number)
14	check digit
15	right guard

The left guard and right guard characters are identical and can be printed using an asterisk or left and right parentheses. The center guard character may be printed using a hyphen or a vertical "pipeline" symbol.

The bar pattern for digits in the left half of the barcode is different from the pattern used on the right side. In the fonts created by the generator, left-side digits are represented in the normal way by the characters 0 through 9. Right-side digits are represented by the letters A through J. For a programmer, encoding the right-side digits is simply a matter of adding the numeric value of each digit to the ASCII value for the character A (41).

Left	*Right*
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
9	J

Let's say we wish to encode the number 0-00123-45678-4 (dashes are shown for clarity). Each of the following examples would produce the same result:

```
*000123-EFGHIE*
(000123-EFGHIE)
*000123|EFGHIE*
```

The programmer must also calculate the value of the check digit. The sample code shown on the following page calculates the check digit and formats the complete string.

Format UPC-A with C

(source file in Sample\UPCA.c)

```
static char upc_string[20];

char *format_upc(char *instring)
{
     char  *cpin, *cpout;
     int   check;
     int   i;

     memset(upc_string, 0, sizeof(upc_string));
     check = upc_check(instring);          // calc check digit
     cpin = instring;                      // point at instring
     cpout = upc_string;                   // point at outstring
     *cpout++ = '*';                       // insert left guard
     for (i = 1; i <= 6; i++)              // insert first 6 digits
          *cpout++ = *cpin++ - '0';
     *cpout++ = '-';                       // insert center guard
     for (i = 1; i <= 5; i++)              // insert next 5 digits
          *cpout++ = 'A' + (*cpin++ - '0');// remember, A-J
     *cpout = '*';                         // insert right guard
     return(upc_string);                   // return to caller
}

int upc_check(char *instring)
{
     int   mysum, mycheck;
     char  *cp;

     mysum = 0;                 // start with a zero
     cp = instring;             // point at start of string
     while (*cp != '\0') {      // go until end of string reached
          mysum += *cp - '0';   // add numeric value of character
          cp += 2;              // go to next odd-numbered position
     }
     mysum = mysum * 3;         // multiply sum so far by 3
     cp = instring;             // start again at beginning
     cp++;                      // move to first even position
     while (*cp != '\0') {      // go until end of string
          mysum += *cp - '0';   // add numeric value of character
          cp += 2;              // go to next even position
     }
     mycheck = 10-(mysum % 10); // calculate Modulo 10
     if (mycheck == 10)         // force result of 10 to 0
          mycheck = 0;
     return(mycheck);           // done - return the result
}
```

Format UPC-A with Basic

(Sample\UPCA.bas on release disk)

```
Function format_upc(instring As String) As String
     Dim check As Integer
     Dim i As Integer
     Dim MyString as String
     Dim MyChar as Integer

     MyString = "*"                              ' left guard
     MyString = MyString & Left$(instring, 6)    ' first 6 digits
     MyString = MyString & "-"                   ' center guard
     for i = 7 to 11                             ' next 5 digits
          MyChar = Val(Mid(instring, i, 1))      ' get digit
          MyChar = MyChar + Asc("A")             ' offset to ABC...
          MyString = MyString & Chr(MyChar)      ' add to string
     next i
     MyChar = upc_check(instring) + Asc("A")     ' check digit
     MyString = MyString & Chr(MyChar)           ' add to string
     MyString = MyString & "*"                   ' right guard
     format_upc = MyString                       ' return to caller
End Function

Function upc_check(instring As String) As Integer
     Dim MySum As Integer
     Dim i As Integer

     mysum = 0                         ' start with a zero
     for i = 1 to 11 Step 2            ' add digits in odd positions
          mysum = mysum + Val(Mid(instring, i, 1))
     next i
     mysum = mysum * 3                 ' multiply result by 3
     for i = 2 to 10 Step 2            ' add digits in even positions
          mysum = mysum + Val(Mid(instring, i, 1))
     next i
     MySum = MySum Mod 10              ' calculate Modulo 10
     MySum = 10 - MySum                ' subtract from 10
     if MySum = 10 then                ' force 10 result to 0
          MySum = 0
     endif
     upc_check = MySum
End Function
```

EAN-13

EAN-13 is the retail barcode used everywhere in the world outside North America. While the first digit in a UPC-A code defines the “number system”, the first 2 or 3 digits in an EAN-13 code identify the country in which the code is registered.

The EAN-13 symbol is almost identical to the UPC-A symbol but encodes 1 extra digit (a total of 13 digits) into the same number of bars and spaces. The first (left-most) digit in an EAN-13 data string is not explicity printed as a separate character in the barcode. Instead, EAN-13 takes advantage of the fact that any of the individual barcode characters can be printed using even or odd parity; the first character is encoded in the even/odd parity pattern in the six left-hand digits of the barcode.

So what does parity mean? Each EAN character is made up of 7 modules which are combined to form 2 bars and 2 spaces. If we use “0” to represent a white module and “1” to represent a dark module, “0001101” would indicate a space (3 modules wide), a bar (2 modules wide), a space (1 module wide, and a bar (1 module wide). This is odd parity, because the total number of dark modules (3) is odd. This particular pattern encodes a zero, which can also be printed with even parity as “01000111”.

To format a string for printing as an EAN-13 symbol, begin with a start code (represented by an asterisk). The next step is to use the first character in the input string to decide what parity pattern to use. Number set A uses odd parity, and number set B uses even parity.

1st Digit	*Number Set to use*
0	AAAAAA
1	BBABAA
2	BABBAA
3	ABBBAA
4	BBAABA
5	BAABBA
6	AABBBA
7	BABABA
8	ABBABA
9	ABABBA

For example, if the first digit in the data string is 4, then we should use the pattern BBAABA. The second digit in the data string (the first digit to be printed in the barcode) and the one after it should be printed using Number Set B; the 4th and 5th digits in the data string should use Number Set A; the 6th digit should use Number Set B; and the 7th digit should use Number Set A.

The ten characters for Number Set A begin at “0” in the font, and Number Set B begins at “K”. To print a character using Set A, add the numeric value of the digit to the ASCII value for “0”:

```
CHR(ASC(“0”) + VAL(mychar))        ... Print mychar using set A
CHR(ASC(“K”) + VAL(mychar))        ... Print mychar using set B
```

After digits 2 through 7 from the input string have been added to the finished string, add the center guard bars using a vertical pipe symbol (|) or a hyphen (-). Print the remaining characters from the input string (8 through 12) using Character Set C, which starts in the font at “A”:

```
CHR(ASC(“A”) + VAL(mychar))        ... Print mychar using set C
```

The next step is to calculate the checksum. Starting from the right end of the input data string, add the numeric values of the digits in the even-numbered positions; the right-most character is in position 1, so start with the second character from the right. Multiply the sum by 3. Then start back at the right end with the character in position 1 and add the numeric values of the digits in the odd-numbered positions into our sum.

Finally, perform a Modulo 10 division on the sum (divide by 10 and use the remainder). Subtract the result from 10; if the answer is 10, change it to zero. This is the checksum. Add this character to the finished barcode string using Number Set C. Finish the string with a guard bar pattern (asterisk). For example, let's print an EAN-13 barcode using the string 719954678901.

1. Start the output string with a guard bar pattern: "*"

2. The first digit in our string is a 7. Using our chart on the preceding page, we will use a pattern of number sets BABABA. Add the next 6 digits to the output string:

Input Digit	*Num Set*	*Compute Character value*
1	B	CHR(ASC("K") + VAL("1"))
9	A	CHR(ASC("0") + VAL("9"))
9	B	CHR(ASC("K") + VAL("9"))
5	A	CHR(ASC("0") + VAL("5"))
4	B	CHR(ASC("K") + VAL("4"))
6	A	CHR(ASC("0") + VAL("6"))

3. Insert the center guard bar pattern: "-"

4. Add the remaining digits using Number Set C:

Input Digit	*Num Set*	*Compute Character value*
7	C	CHR(ASC("A") + VAL("7"))
8	C	CHR(ASC("A") + VAL("8"))
9	C	CHR(ASC("A") + VAL("9"))
0	C	CHR(ASC("A") + VAL("0"))
1	C	CHR(ASC("A") + VAL("1"))

5. Calculate the checksum:
   a. Starting from the right, add up the digits in the even-numbered positions:
      7 + 9 + 5 + 6 + 8 + 0 = 35
   b. Multiply the sum by 3:
      35 * 3 = 105
   c. Starting from the right, add the digits in odd-numbered positions to the sum:
      105 + 1 + 9 + 7 + 4 + 9 + 1 = 136
   d. Perform a Modulo 10 division:
      136 Mod 10 = 6
   e. Subtract the result from 10:
      10 - 6 = 4
   f. If the result is 10, change it to zero. In this case, final result is 4

6. Add the checksum to the output string using Number Set C:
   CHR(ASC("A") + VAL("4"))

7. Add the right guard bar pattern: "*"

The sample file ModuleEAN13.bas contains the following Visual Basic functions that will perform the formatting functions.

```
Function format_ean(instring As String) As String
    Dim check As Integer
    Dim i As Integer
    Dim outstring As String
    Dim mychar As String
    Dim myset As String
    Dim setstr As String
    Dim mystring As String

    '
    '   Make sure that we have been given 12 digits
    '
    If IsNull(instring) = True Then
        mystring = "000000000000"
    ElseIf Len(instring) < 12 Then
        mystring = ""
        For i = 1 To (12 - Len(instring))
            mystring = mystring & "0"
        Next i
        mystring = mystring & instring
    ElseIf Len(instring) > 12 Then
        mystring = Mid$(instring, 1, 12)
    Else
        mystring = instring
    End If

    '
    '   mystring now contains the input data string.
    '   Determine what pattern of EAN character sets should be
    '   used based on the leftmost character.  The choice of
    '   character set pattern encodes the value of the first digit
    '
    setstr = ean_charset(Mid$(mystring, 1, 1))

    '
    '   Start with the left guard bar pattern
    '
    outstring = "*"

    '
    '   Insert the next 6 digits, looking up the correct character
    '   based on the EAN character set for each position.  We do not
    '   insert the first character; it is encoded in the parity
    '   pattern of digits 2 through 7
    '
    For i = 2 To 7
        mychar = Mid$(mystring, i, 1)
        myset = Mid$(setstr, i - 1, 1)
        outstring = outstring & ean_char(mychar, myset)
    Next i

    '
    '   Insert the center guard
    '
    outstring = outstring & "|"
```

continued on next page...

```
    '
    '   Now insert the next 6 digits.
    '
    For i = 8 To 12
        mychar = Mid$(mystring, i, 1)
        outstring = outstring & ean_char(mychar, "C")
    Next i

    '
    '   Calculate the check digit and append it to the string
    '
    check = ean_check(mystring)
    outstring = outstring & ean_char(Chr(check), "C")
    '
    '   Add the right guard bar pattern and return to caller
    '
    outstring = outstring & "*"
    format_ean = outstring
End Function

Function ean_check(instring As String) As Integer
    Dim cp As Integer
    Dim sum As Integer
    Dim ck As Integer

    '
    '   Initialize the sum to zero and add up the values of
    '   characters in even positions starting from the right
    '
    sum = 0
    For cp = 12 To 1 Step -2
        sum = sum + Val(Mid$(instring, cp, 1))
    Next cp

    '
    '   Multiply the result by 3, then add in the values of
    '   characters in the odd positions
    '
    sum = sum * 3
    For cp = 11 To 1 Step -2
        sum = sum + Val(Mid$(instring, cp, 1))
    Next cp

    '
    '   Do a Modulo 10 division on the result and subtract it
    '   from 10.  If the result is 10, set it to zero.  This is
    '   the final checksum.
    '
    ck = 10 - (sum Mod 10)
    If ck = 10 Then
        ck = 0
    End If
    ean_check = ck
End Function
```

```
Function ean_charset(leftchar As String) As String
    Dim Sets() As Variant
    Dim MyVal As Integer

    '
    '   This array defines the parity pattern to be used for digits
    '   2 through 7.  The pattern encodes the value of the first
    '   character.
    '
    Sets = Array("AAAAAA", "BBABAA", "BABBAA", "ABBBAA", "BBAABA", "BAABBA", "AABBBA",
"BABABA", "ABBABA", "ABABBA")

    '
    '   Get the numeric value of the first (left-most) character and
    '   return the correct string based on its value.
    '
    MyVal = Val(leftchar)
    If MyVal < 0 Or MyVal > 9 Then
        MyVal = 0
    End If
    ean_charset = Sets(MyVal)
End Function

Function ean_char(datachar As String, charset As String) As String
    Dim i As Integer

    '
    '   This function returns the character that should be printed
    '   based on the value of the character and the particular
    '   character set that should be used (A, B, or C)
    '
    i = Val(datachar)
    If charset = "A" Then
        i = Val(datachar) + Asc("0")
    ElseIf charset = "B" Then
        i = Val(datachar) + Asc("K")
    ElseIf charset = "C" Then
        i = Val(datachar) + Asc("A")
    End If
    ean_char = Chr(i)
End Function
```

Interleaved 2 of 5

(Sample\ITF.bas on release disk)

Interleaved 2 of 5 is a compact, numeric-only barcode that encodes a pair of digits on each barcode character. For example, "35" is encoded as a single character; when this character is read by the barcode scanner, it is separated into two digits before transmission to the computer.

The first character in the Interleaved 2 of 5 PCL font represents the digit pair "00", and it is located in the font where you would expect to find the zero (0) character (ASCII character value 48). By adding the numeric value of the desired digit pair to the ASCII value of the zero character, you will obtain the ASCII value of the character that you should print.

Calculating the Checksum

An Interleaved 2 of 5 barcode may also include a checksum, which is calculated using the Modulo 10 method. Since the final barcode string (including the checksum) must have an even number of digits, an extra character (typically a leading zero) may have to be added to the original data. The checksum is calculated on the string of digits before they are divided into pairs; the start character is not included in the calculation:

1. Starting with the digit in position 1 (the left-most digit), add the values of all the digits in the odd-numbered character positions (1, 3, 5, etc.).

2. Multiply the result of Step 1 by 3.

3. Add the values of the digits in the even-numbered positions to the result of Step 2.

4. Perform a Modulo 10 division on the result of Step 3; that is, divide the result of Step 3 by 10 and take the remainder.

5. Subtract the result of Step 4 from 10; if this result is 10, change it to zero. This is the checksum which should be appended to the end of the data string.

The following sample function will return a formatted string for printing as an Interleaved 2 of 5 barcode (the source code is included on the Font Kit release disk):

```
Function FmtITF(InString As String, Checksum As Boolean) As String
      Dim i As Integer
      Dim MySum As Integer
      Dim WorkString As String
      Dim FinishedString As String
      Dim PairVal As Integer
      Dim CharVal As Integer
      Dim PairString As String

      '
      '     check for illegal alpha characters
      '
      If (IsNumeric(InString) = False) Then
            FmtITF = ""
            Exit Function
      End If
```

```
     '
     '     insure even number of digits in final string
     '
     i = Len(InString) Mod 2
     If ((Checksum = False And i <> 0) Or (Checksum = True And i=0)) Then
          WorkString = "0" & InString
     Else
          WorkString = InString
     End If
     '
     '     calculate checksum if requested
     '
     If Checksum = True Then
          MySum = 0
          '
          '     Add up all digits in the odd-numbered positions
          '     then multiply result by 3
          '
          For i = 1 To Len(WorkString) Step 2
               MySum = MySum + Val(Mid(WorkString, i, 1))
          Next i
          MySum = MySum * 3
          '
          '     Add in all digits in even-numbered positions
          '
          For i = 2 To Len(WorkString) Step 2
               MySum = MySum + Val(Mid(WorkString, i, 1))
          Next i
          '
          '   Finish the calculation
          '
          MySum = MySum Mod 10
          MySum = 10 - MySum
          If MySum = 10 Then
               MySum = 0
          End If
          WorkString = WorkString & Format(MySum)
     End If
     '
     '     Build the finished string by mapping pairs of digits
     '     to correct character positions in the font
     '
     FinishedString = "("
     For i = 1 To Len(WorkString) Step 2
          PairString = Mid(WorkString, i, 2)
          PairVal = Val(PairString)
          CharVal = PairVal + Asc("0")
          FinishedString = FinishedString & Chr(CharVal)
     Next i
     FinishedString = FinishedString & ")"   ' stop character
     FmtITF = FinishedString
End Function
```

Postnet

The Postet barcode is the row of tall and short bars that often appear below or above an address on a letter. Postnet encodes the Zip Code so that it can be read by automatic sorting equipment. A Postnet barcode can include the 9-digit Zip+4 code or the 11-digit Delivery Point Code, which is the same as Zip+4 with two extra digits to define the destination in more detail. Each digit is represented by five bars, two tall and three short, and the complete barcode is constructed as follows:

- Start character (one tall “frame bar” or “guard bar”) represented by letter L or l (left)
- Numeric data (5 digits for Zip Code, 9 digits for Zip+4, or 11 digits with Delivery Point Code)
- Check Digit
- Stop character (one tall “frame bar” or “guard bar”) represented by letter R or r (right)

For example, the following text string will produce a complete Postnet barcode:

```
L1234567895R
```

The check digit is used to insure accuracy when the barcode is scanned. The check digit is calculated using the 5, 9, or 11 digits of Zip Code data and must be calculated for each barcode. When the barcode is read, the scanning equipment performs the same calculation and compares its result with the check digit that was read from the barcode. If the two do not match, the scanner knows that there is something wrong with the data and can eject the letter for manual sorting.

Here is the general method for calculating the checksum. Starting from the left, sum all of the digits in the barcode. Using the example above:

```
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45
```

For the next step, we need the units column of the sum. You can extract this information using Modulo 10 division, which is the same as dividing by 10 and taking the remainder.

```
Basic:     45 MOD 10 = 5
C:         45 % 10 = 5
```

Subtract this result from 10 to obtain the check digit; if the final result is 10, change it to zero. In the example above, the check digit would be 5. Here is a bit of sample code written in Basic:

```
ZipString = “123456789”
Sum = 0
FOR x = 1 to LEN(ZipString)
     Sum = Sum + VAL(MID$(ZipString, x, 1))
NEXT x
Check = 10 - (Sum MOD 10)
IF Check = 10 THEN
     Check = 0
ENDIF
PrintString = “L”+ZipString+Str$(Check)+”R”
```

Here is the same function in C:

```
char  ZipString[20], PrintString[20], *cp;
int   Sum, Check;

strcpy(ZipString, “123456789”);
Sum = 0;
cp = ZipString;
while (*cp != ‘\0’)
     Sum += (*cp++) - ‘0’;
Check = 10 - (Sum % 10);
if (Check == 10)
     Check = 0;
sprintf(PrintString, “L%s%1dR”, ZipString, Check);
```

Planet Code

The Planet Code is a variation of the Postnet code used by the United States Postal Service. The height of the bars is inverted, so each Planet character has three tall bars and two short ones. The left and right frame bars are the same as Postnet, and the checksum calculation method is the same.

Planet Code is used for four applications, each with 12 digits:

- Mailpiece rate, pre-sort bureau origin, and customer identification
- Computer on-line notification and identification of person returning reply mail
- Sorting of foreign mail
- Automated address correction requests